

REMARKS

The present amendment is filed concurrently with a Request for Continued Examination (RCE).

Claims 25-27, 29, 80-81 and 83 stand rejected as being unpatentable over Helbig in view of England. Reconsideration and withdrawal of these rejections are respectfully requested.

The Examiner will note that claims 25 and 80 have each been amended to recite:

providing and installing a trusted verification driver in the gaming machine, the trusted verification driver being independent of the operating system;

disabling, using the trusted verification driver, all interrupts generated by the gaming machine;

performing a verification of components of the operating system against a trusted reference using the trusted verification driver while all interrupts are disabled and preventing further operation of the gaming machine when the verification of the components of the operating system fails;

Support for the disabling step may be found, for example at page 26, line 19 to page 27, line 17, page 28, line 10 to page 30, line 3 and page 31, line 10 to page 32, line 3 and corresponding figures.

New claims 143-145 depend on claim 25. Claim 143 recites a step of enabling, using the trusted verification driver, all interrupts generated by the gaming machine. Such re-enabling may be carried out before the checking step (claim 144, Fig. 8) that checks code signature of the downloaded software module or after the checking step (claim 145, Fig. 9). New claims 146-148, dependent on claim 80, recite similar steps and functionality. The Large Entity fee for the six new dependent claims may be debited from the undersigned's Deposit Account, number 50-3159.

The Helbig-England combination does not teach or suggest any such methods or gaming machines. Helbig discloses three methods for halting the I486 (the PC's processor):

EXERTING CONTROL OVER (DEACTIVATING) THE I486

There are alternate ways of halting SEPB operation of the I486 to allow the RISC processor to execute the proper portion of its SEPB firmware. The preferred embodiment is to intercede in a memory bus cycle. The CPU begins a bus cycle by having bus access, placing an address on the address bus lines and sending out the ADS# signal. The I486 then awaits the return of the "ready" signal that signifies the conclusion of the bus cycle. The "ready" signal is sent to the CPU by the addressed PC unit to indicate either that the data the CPU has requested is ready for it to accept or that the data sent by the CPU has been accepted by the addressed unit. If the "ready" signal is not received by the CPU it will wait forever for it. For the SEPB to stop the CPU, the SEPB firmware being executed by the CPU contains a "write data" operation to send data to the multiprocessor logic controller. To stop the CPU's execution of its SEPB firmware at this point the multiprocessor logic controller simply does not return the "ready" signal to the CPU. Instead the logic controller starts the RISC processor's execution of its SEPB firmware by restarting the generation of the RISC clock and causing the I486 to release bus and control signals.

In this manner the goal of having either the CPU executing its SEPB firmware or the RISC processor executing its firmware, but not both simultaneously, during periods of SEPB operation is achieved. When the multiprocessor logic controller knows that the CPU is waiting for a "ready" signal but it wants the RISC processor to execute its SEPB firmware the multiprocessor logic controller will generate, and send to the CPU, a "hold" signal to get the CPU to release control of both the Processor Address Bus and the Processor Data Bus by the CPU putting its bus driver circuits into the High impedance state. Then, with the Processor Address Bus and the Processor Data Bus free the RISC processor can use them whenever it wants to transfer data between it and other parts of the SEPB or PC.

Later, when the execution of the RISC processor SEPB firmware reaches a point where it is to be stopped and the execution of the CPU SEPB firmware is to be restarted, the RISC processor performs a similar "write data" operation to send a signal to the multiprocessor logic controller to perform the operations necessary to make the switch. When the multiprocessor logic controller detects this special "write data" operation it stops the operation of the RISC processor and releases the CPU to allow it to continue its execution of its SEPB firmware by simply stopping the generation of the clock signal for the RISC processor, stopping the generation of the "hold" signal and ceasing sending it to the CPU, and generating a "ready" signal and sending it to the CPU.

An alternate implementation to control a CPU, is to stop the clock of the CPU as it does the RISC processor clock. However, this approach is somewhat limited because it cannot be used with CPU's which have an on-chip phased locked loop to increase its internal clock speed. Such phased locked loops, used for clock rate multiplication, continue to run for a while after the external clock has been stopped and take a non zero amount of time to reach its full operation state after the external clock has been started.

Yet another alternate embodiment to exert control over a CPU is to halt the CPU as is done in a multiple processor master slave arrangement where the master CPU halts the slave CPU. The CPU may be

also be controlled by intercepting one interrupt signal and substituting another, or substituting a different interrupt vector, or a interrupt address for the control signal. Other ways to control a CPU include intercepting a write strobe signal, intercepting a read strobe signal or, as described above, by intercepting a data ready signal.

As used herein, a write strobe is any one of the type of control signal in which a CPU indicates that the information output to a bus (data or address) is to be written elsewhere. An example is the address strobe signal, ADS#, which is the start signal for the processor bus. A read strobe is any one of the type of control signal in which a CPU indicates that the information input from a bus (data or address) is to be read. A data ready signal is any one of the type of control signal to indicate that information (data or address) has been placed on or read from a data or address bus.

Yet other ways to assert control over the CPU is to intercept address and/or data lines. For example, intercepting data lines permits the substitution of different executable code than would be normally provided to the CPU from the addressed memory space. Intercepting address lines would force the CPU to alternative memory space for executable program or data. Col. 10, line 37 to Col. 11, line 55.

The first method, therefore, is to withhold sending the bus "ready" signal, for which the CPU will wait forever. The second method is to stop the processor's clock, which does not work for CPUs having a PLL which continues operation after the clock has been stopped. The third method detailed above involves intercepting an interrupt signal generated by the CPU and substituting another interrupt signal for the intercepted interrupt or substituting another interrupt vector or interrupt address for the control signal. Notwithstanding the fact that Helbig says that there are only three methods, Helbig goes on to also discloses that the PC's i486 processor may be halted by intercepting a write strobe signal, intercepting a read strobe signal, by intercepting a data ready signal, by intercepting address and/or data lines.

With regard to interrupts, Helbig et al. teach to intercept interrupts and re-direct interrupt vectors, which necessarily entails that the interrupts are NOT disabled, merely intercepted and redirected. Therefore, Helbig et al. do not teach or suggest disabling all interrupts, as required by the claimed embodiments. In turn, therefore, even if Helbig et al. were to be combined with the England, relied upon for downloaded software modules, the claimed embodiments would remain

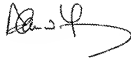
untaught and unsuggested by the applied reference. Such a combination would still allow the PC's processor to continue generating interrupts, which interrupts would be intercepted and the interrupt vectors redirected and the downloaded software module would be verified using an un-trusted base.

Moreover, such a combination would not teach or suggest the step of "performing a verification of components of the operating system against a trusted reference using the trusted verification driver while all interrupts are disabled and preventing further operation of the gaming machine when the verification of the components of the operating system fails", as claimed herein.

As the claimed embodiments require steps that are not taught or suggested in the applied combination, reconsideration and withdrawal of the obviousness rejections applied to the claims are, therefore, respectfully requested.

If any unresolved issues remain, the Examiner is respectfully invited to contact the undersigned attorney of record at the telephone number indicated below, and whatever is required will be done at once.

Respectfully submitted,



Date: August 27, 2010

By:

Alan W. Young
Attorney for Applicants
Registration No. 37,970

YOUNG LAW FIRM, P.C.
4370 Alpine Rd., Ste. 106
Portola Valley, CA 94028
Tel.: (650) 851-7210
Fax: (650) 851-7232